Marcin Petrowicz
Uniwersytet Jagielloński

Game design patterns
from the university to design studio

In 1977 Austrian architect Christopher Alexander published his best known book *A Pattern Language*[1]. The concept presented nearly 40 years ago, inspired computer scientists and programmers alike, leading to the creation of design pattern language for object-oriented programming. However, despite its popularity the project was not recognized and accepted as a practical tool for architects. Years later, when the video game industry was in need of a formal language to describe the craft, an idea to use pattern language appeared. Two game researchers – Staffan Björk and Jussi Holopainen conceived a collection of design patterns[2]. But, when it inspired game studies researchers for some time, the ideadid not gain the attention of the wide game industry community. Despite the partial failure of adapting pattern language to video game design some researchers, like Joris Dormans[3], and designers, like Raph Koster[4], work towards the creation of a formal pattern-like language of gameplay.

The following paper examines different concepts of design pattern languages from the original one in architecture, through the most popular pattern concepts in object-oriented programming to Björk's and Holopainen's work in the field of video game design. The focal point of analysis and comparison will be the languages' foundations and assumptions, single design pattern structure, and level of abstraction. Beside the internal structure of each pattern language, the reception

---

1 C. Alexander, S. Ishikawa, M. Silverstein, *A Pattern Language: Towns, Buildings, Construction*, New York 1977.

2 S. Björk, J. Holopainen, *Patterns in Game Design*, Hingham 2005.

3 J. Dormans, *Engineering Emergence. Applied Theory for Game Design*, 2012 [on-line:] http://www.jorisdormans.nl/pdf/dormans_engineering_emergence.pdf [30.04.2016].

4 R. Koster, Grammar of Gameplay. Game Atoms: Can Games Be Diagrammed?, 2005 [on-line:] http://www.raphkoster.com/gaming/atof/grammarofgameplay.pdf [30.04.2016].

in the professional community will also be examined. The central issues addressed in this paper are the characteristics of specific pattern languages that made them popular and useful or led them to be rejected by the industry. After this comparison, both shortcomings of Björk's and Holopainen's concept and possible solutions for similar future theories will be examined.

Patterns in architecture

Born in 1936 in Vienna, Christopher Alexander is mainly an architecture theorist and practitioner. During his education at Cambridge and Harvard universities, he was also interested in physics and mathematics, from which he obtained a Master's degree. His interests in disciplines other than architecture allowed him to work in a very diverse range of academic contexts, from transportation theory and computer science to cognitive studies. Alexander's study background of different systems must have been the inspiration for the creation of generative grammar for architecture – the pattern language.

Published in 1977 by Alexander and his collaborators – Sara Ishikawa and Murray Silverstein, *A Pattern Language: Towns, Buildings, Construction* is the second in a series of books released by *Center for Environmental Structure*. It was followed by *The Timeless Way of Building*[5], which is the philosophical and aesthetic introduction to the pattern language, even though it was released in 1979. The "timeless way", mentioned in the title, is a method that once grasped and understood can be implemented to produce beautiful buildings in every style, culture or place it is built in. The starting point for the creation of patterns is "a central quality which is the root criterion of life and spirit in a man, a tow, a building or a wilderness. This quality is objective and precise, but it cannot be named"[6]. The first book in the series, that actually describes the pattern language in action, is *The Oregon Experiment*[7] that introduces the process of campus community planning at the University of Oregon. After series of student protests, university authorities, in a conciliatory gesture towards the community, employed the radical architecture professor from University of Berkeley to solve the problem . Alexander's task was to create a process in which the community could design its own urban environment by itself. He prepared the list of good practices and patterns to facilitate the process of community planning as a tool for making this seemingly anarchist act of creation a bit more sensible.

The structure of the pattern language

The Pattern Language consists of more than 250 patterns divided into three categories – towns, building and construction, ranging from general patterns as 26.

5 C. Alexander, *The Timeless Way of Building,* New York 1979.

6 Ibidem, p. 19.

7 Idem, *The Oregon Experiment,* New York 1975.

Life Cycle[8] to more particular solutions as 201. Waist-High Shelf[9]. Every pattern is presented as a pair of commonly occurring problem and a universal solution that meets the requirements of the timeless way of building, one that brings us closer to the quality without a name. For example pattern number 201 describes the problem of "daily »traffic«" of the objects which are handled most often that occurs in every house and every workplace. "Unless such things are immediately at hand (...) [MP – they] are forgotten, misplaced"[10]. To transform this part of our daily routine into the form of architecture the author proposes building waist-high shelves in every main room where people spend their time.

Looking at the structure of any given pattern we observe that its starting point is usually some kind of human behaviour, need or characteristic. Then all the features and forces acting on the object in question are discussed. Afterwards the best configuration of parameters is chosen in a way to lead to the desired qualities described in *Timeless Way of Building*. Next is "solution – the heart of the pattern – which describes the field of physical and social relationships which are required to solve the stated problem"[11]. Finally, every pattern ends with a list of other patterns related to the described issue. Another interesting feature of Alexander's language is the supposed reliability of every pattern. Author, being aware of his own shortcomings, marked the patterns with two, one or no asterisks. Two asterisks indicate that the pattern summarizes "a *property* common to *all possible ways* of solving the stated problem"[12], one asterisk marks patterns whichwere designed into this general direction of invariant solution, but can be improved, while no asterisk informs the reader that the described solution is only one specific way of solving the problem and it can be improved upon.

Reception

Despite Christopher Alexander is a successful and influential architect, his pattern language is not as important concept in designing buildings process as in the world of software design. Nikos A. Salingaros, a proponent of Alexander's patterns, observes that "pattern languages encapsulate human experience, and help us cope with complexity in our environment"[13]. However the language "has been appreciated by only a few practitioners" and "because of misunderstandings, they [MP – patterns] have not played a wide role in architectural design"[14]. The question

8 C. Alexander, S. Ishikawa, M. Silverstein, op. cit., pp. 139–145.

9 Ibidem, pp. 922–923.

10 Ibidem, p. 922.

11 C. Alexander, S. Ishikawa, M. Silverstein, op. cit., p. xi.

12 Ibidem, p. xiv.

13 N. A. Salingaros, *The Structure Of Pattern Languages*, "Architectural Research Quarterly" 2000, vol. 4, no. 2, p. 18.

14 Ibidem.

is: what makes probably the most successful book on architectural concept more popular amongst layman than professional architects[15]?

As it was presented by Kimberly Dovey in the article on *The Pattern Language and its Enemies*, the critique of Alexander's language can be grouped in four categories: political, ideological, epistemological and aesthetical[16]. The first two acknowledge that the potential drawbacks of Pattern Languages are anchored in theoretical discourses of the 1960' – Lévi-Strauss' structuralism and counterculture's fascination with the East. The Taoist "quality without a name" strictly opposes the Western philosophy with its dualism and empiricism, while many patterns openly oppose consumerism and capitalism[17]. The epistemological criticism points to the questionable assurance that pattern solutions are absolutely right (at least with two asterisks patterns). The last type of opponents, as presented by Kimberly Dovey, raise the issue that the pattern language is permeated by 'form follows function' mindset, which is a principle of specific movement namely modernist architecture[18]. While this could have been a popular concept in the first half of the 20th century, it has lost its popularity in the second half, especially amongst postmodernist architects. What is more, most patterns go against economic determinants of the building industry. It is difficult, or even impossible, to accord the goal of achieving best living quality by means of patterns with the goal of increasing income for the developer's company. While there are different arguments against those criticisms[19], the main answer to the opponent's concerns is the renunciation of Alexander's absolutism. If one would read *Pattern Language* as a programme of specific style or fashion of designing, then that simple act of relativism would make some patterns much more acceptable. On the other hand, the absolutism and universal idea behind the concept is exactly what makes it to be an inspiring and convincing method.

Patterns in software design

Ten years after the publication of *The Pattern Language* it caught attention of two prominent software engineers – Kent Beck and Ward Cunningham. Beck is the creator of extreme programming – a software development method maintained in agile paradigm and a pioneer of a commercial application of the Smalltalk programming language. He later worked with Erich Gamma, creating JUnit –

---

15 See: T. Erickson, *Lingua Francae for Design: Sacred Places and Pattern Languages*, "Proceedings of the ACM Conference on Designing Interactive Systems 2000" New York 2000 [on-line:] http://www.pliant.org/personal/Tom_Erickson/LinguaFranca_DIS2000.html [30.04.2016].

16 K. Dovey, *The Pattern Language and its Enemies*, "Design Studies" 1990, vol. 11, p. 4.

17 C. Alexander, S. Ishikawa, M. Silverstein, op. cit., p. 393.

18 See: L. Sullivan, *The Tall Office Building Artistically Considered*, "Lippincott's Monthly Magazine", March 1896.

19 See: T. Turner, *City as landscape: A Post Post-Modern View of Design and Planning,* Abingdon 1996, pp. 24–27.

an important testing framework for Java. Cunningham is also an extreme programming pioneer and the programmer who developed the first wiki. Actually, the purpose of the predecessor of Wikipedia was to "create an environment where we might link together each other's experience to discover the pattern language of programming"[20]. Beck and Cunningham started exploring the possibilities of adapting pattern design methods to software development in 1987, when they presented the idea of *Using Pattern Languages for Object-Oriented Programs*[21] at the OOPSLA conference. Beck and Cunningham continued their work on software design patterns for a few years more, but it was not until the 1994 that the concept gained popularity also in computer science and software development. In 1994 the so-called *Gang of Four* – Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides published the book entitled *Design Patterns: Elements of Reusable Object-Oriented Software*[22]. The first part of the publication is devoted to good practices in object-oriented programming, then it introduces the concept of software patterns. The second half of *Design Patterns…* is devoted to the description of 23 classic software design patterns. Probably, it can be perceived as the most important publication that made the pattern language concept famous outside the architecture and design fields.

Software patterns specifics

Similarly to its prototypical architectural counterparts the software development patterns are the best known solutions to typically occurring problems. The patterns are not specific to a particular programming language, but, as the book title states, they were developed with an object-oriented paradigm in mind. The structure of the software design patterns is a developed version of the architectural patterns. Each entry consists of a name, intent, other names of the pattern, motivation for using, the situation in which the pattern can be applied, the structure, participants – classes and objects participating in a pattern, collaborations between the participants, consequences, characteristics of implementation, sample code, known uses and related patterns. However, similar to Alexander's patterns, software design patterns had some differences occurring as a result of being born in a completely different design system.

Gang of Four patterns were prescriptive and proven to be working in much stronger sense than architectural patterns. It was not a matter of ideological choice, but a characteristic of working with logical and quantifiable algorithms.

---

20 B. Venners, *Exploring with Wiki. A Converstaion with Ward Cunningham, Part I,* 2003 [on-line:] http://www.artima.com/intv/wiki.html [30.04.2016].

21 K. Beck, W. Cunningham, *Using Pattern Languages for Object-Oriented Programs,* Object-Oriented Programming, Systems, Languages & Applications conference 1987 [on-line:] http://c2.com/doc/oopsla87.html [30.04.2016].

22 E. Gamma, R. Helm, R. Johnson, J. Vissides, *Design patterns: Elements of reusable object-oriented software,* Reading 1995.

Being effective – meaning faster, or less memory consuming – was the goal of every pattern or, as the authors wrote, "design patterns make it easier to reuse successful designs [...], design patterns help a designer get a design »right« faster"[23].

As the software development is a vast field of knowledge no one would try, as Alexander and his collaborators did, to build a complete or even nearly finished list of patterns. That is why Cunningham created wiki to accumulate patterns and experiences of many programmers, and that is why the book by *Gang of Four* presents only 23 patterns. Itss goal was to show and describe a framework for software design which can be developed by other computer scientist afterwards.

Reception in software development

The concept of design patterns in software development and computer science was very well received, as the popularity of the aforementioned book shows. It was adapted to many more specific domains of software development and different kind of programming languages. Patterns have been used practically in many professional contexts because for those, who master them, they offer great rewards. They speed up the development, give effective, reliable and proven solutions and what is more, improve code readability. Although software design patterns have its critics. Most of them concentrate on the fact that the problems presented in the patterns could be easily solved or omitted by simply changing the programming environment[24]. Despite some criticism there are no significant arguments to support the view that creating patterns for software development is a professionally worthless endeavour.

Patterns in game design

Beginning of the 21st century was marked by a major shift in the game industry. Increased access to broadband internet connection, a new type of video game motion controller and transformation of advanced mobile phones into smartphones led to, so-called, casual revolution[25]. New, casual genres of games were created for the group of consumers that would never before call themselves gamers (for example *Bejeweled* (PopCap Games, 2001), *Wii Play* (Nintendo, 2006), *Mystery Case Files: Huntsville* (Big Fish Studios, 2005)). The increasing amount of new clients and the expansion of hardcore consumer base both led to a great growth of the video game market. Rapid advancement of the electronic game industry resulted in increased demand for new methods of game development, not only technological, but also theoretical. Requests for game designing tools, theoretical

---

23 Ibidem, p. 12.

24 See: P. Norvig, *Design Patterns in Dynamic Programming,* 1996 [on-line:] http://www.norvig.com/design-patterns/design-patterns.pdf [30.04.2016]; J. Hannemann, G. Kiczales, *Design pattern implementation in Java and AspectJ,* "ACM Sigplan Notices" 2002, vol. 37. no. 11.

25 See: J. Juul, *A casual revolution:* Reinventing *video games and their players,* Cambridge 2009.

language in particular, appeared in 1999, when Doug Church stated that "the primary inhibitor of design evolution is the lack of a common design vocabulary"[26]. In response to that both researchers and practitioners published papers[27] and books[28] which aimed in delivering requested design theories, tools and vocabulary. It was not long before someone proposed to use the designing patterns that were already a familiar concept for many games programmers[29]. The concept was adopted by the game studies and then developed by aforementioned scholars - Staffan Björk and Jussi Holopainen. In their book the researchers present game component framework – a model for structural analysis of a video game, then they proceed to describe the modified concept of design patterns. However, the main part of the publication is a collection of nearly 300 patterns.

Game design patterns specifics

Reflecting upon Christopher Alexander's language and application of patterns to software development Björk and Holopainen implement necessary changes to pattern method. They present three arguments against the problem-solution structure: it "creates a risk of viewing patterns as a method for only removing unwanted effects of a design rather than tools to support creative design work"[30], many identified patterns consist problems that are related to other patterns that describe the solutions and "the effect of introducing, removing, or modifying a game design pattern in a game affected many different aspects of the gameplay, making game design patterns imprecise tools for solving problems mechanically"[31]. In place of the previous structure, authors propose cause and consequence categories describing the conditions for utilizing the pattern and the potential effects on the entire designed system. As a result of those reflections, the basic game designing pattern template consists of: a name, core definition, general description, usage of the pattern, consequences, relations and references. The scope of patterns varies from very general like *Game World*[32] or *Avatars*[33] to more specific

---

26 D. Church, *Formal Abstract Design Tools*, 1999 [on-line:] http://www.gamasutra.com/view/feature/131764/formal_abstract_design_tools.php [30.04.2016]

27 J. P. Zagal, M. Mateas, C. Fernandez-Vara, B. Hochhalter, N. Lichti, *Towards an Ontological Language for Game Analysis* [in:] *Changing Views: Worlds in Play, Selected Papers of DIGRA 2005,* ed. S. de Castell, J. Jenson, Vancouver 2005, pp. 3–14.

28 K. Salen, E. Zimmerman, *Rules of play: Game design fundamentals,* Cambridge 2003.

29 S. Chen, D. Brown, *The Architecture of Level Design*, 2001 [on-line:] http://www.gamasutra.com/resource_guide/20010716/chen_pfv.htm [30.04.2016]; B. Kreimeier, *The Case for Game Design Patterns*, 2002 [on-line:] http://www.gamasutra.com/view/feature/132649/the_case_for_game_design_patterns.php?print=1 [30.04.2016].

30 S. Björk, J. Holopainen, op. cit., p. 34.

31 Ibidem.

32 Ibidem, pp. 55–58.

33 Ibidem, pp. 77–79.

as *Cut Scenes*[34] or *Easter Eggs*[35]. Most of the patterns are well established or at least recognizable concepts in game design. The content of each pattern blends the description with emphasis on relations to other patterns and offers general advice for using each structure in game design.

Reception in game studies and design

The reception of Björk's and Holopainen's work in the game design community was rather reluctant. To this day there is no evidence of a well-known game designed with the help of this concept. A quick search on *Gamasutra,* an important game development web portal founded in 1997, gives few entries related to the subject. On the other hand, it is nearly impossible to examine to what extent the book *Patterns in Game Design* fulfilled the goal of creating and popularizing a common language for game design analysis. The game design pattern framework is the most popular one among the members of the game studies community. Within academic context, the framework was developed and applied to more specific game design issues, including: FPS level design[36], NPC design[37] or even unethical design[38].

Summary

The above comparison of three design pattern methods shows that it was most successful in the industry where the results can be easily measured – namely in programming. Thus, maybe it is not the most appropriate method for more open-ended or "creative" endeavours like a game or architecture design. The first language the Alexander's framework was successful as an idea that inspired many theoreticians, academics and spreaded to other disciplines, but was not applied in its own field. In comparison, the second language is arguably the most popular language. A great deal of the programmer's work is finding solutions for well defined, precise problems, and that is where patterns can be really helpful. It was not only replicated in many specific areas of computer science, but also served as a helpful tool for professional work. The third language left the problem-solution structure in an attempt to cater game designers and answer their needs for the lingua franca. Instead, it was mostly ignored by game developers, but

34 Ibidem, pp. 233–234.

35 Ibidem, pp. 235–236.

36 K. Hullett, J. Whitehead, *Design Patterns in FPS Levels* [in:] *Proceedings of Foundations of Digital Games Conference,* Montreal 2010.

37 Lankoski P., *Character-Driven Game Design – A Design Approach and Its Foundations in Character Engagement,* Jyväskylä 2010.

38 J. P. Zagal, S. Björk S., C. Lewis, *Dark patterns in the design of games,* 2013 [on-line:] http://soda.swedish-ict.se/5588/1/DarkPatterns.1.1.6_cameraready.pdf [30.04.2016].

nonetheless accepted by the research community leading to the development of patterns concept.

What are the reasons for this distribution of recognition among professional groups? Is it mainly due to differences between architecture, programming and game design? Or is it a matter of choosing the right method for each discipline? Another explanation could be that the appreaciation of pattern methods in each discipline depends on the assumptions and theoretical foundations of each design language.

From university halls to design studios

In all three cases pattern languages were developed in academic context, but also a professional adaptation was intended. Christopher Alexander had the possibility of working with his pattern method himself many times during his practice as an architect, notably while planning for the University of Oregon. In comparison, computer science and software development are strongly linked, as in cases of OOPSLA conference, where the software design patterns were presented for the first time. While working on the concept, Beck and Cunningham could not have done this without really getting to the code. Whereas the authors of game design patterns were involved in the development of few game titles, they were primarily game researchers, not designers.

As we can see, the transfer from the university walls to development studio was largely dependent on the discipline context, but it was also related to the discipline specifics. While comparing architecture, software development and game design, what is beyond the scope of this paper, basic differences can be noted. Architecture, as understood in Alexander's book, is an activity aimed at planning the most comfortable and functional building for its users. As a result, austrian architect's pattern language is prescriptive and aims at describing how to achieve quality without a name through designing the physical environment in which we live. In software development efficiency is the key, either in the sense of development speed, memory efficiency of the program or even elegance and the simplicity of the code. For this reason software patterns form a collection of best solutions for popular problems, a collection of accumulated knowledge for developers obtained from more advanced and experienced colleagues. It aims at giving a mechanical resolution of rational software design problem. Moreover, it names these repeatable and recognizable patterns. Game design is comparable to architecture. It is a creative discipline in which the vast variety of project types seemingly excludes the possibility to write down common qualities for all those projects. It can be done in form of Alexander's quality without a name but that is not what Björk and Holopainen have chosen. Instead, they abandoned the prescriptive aspect and concentrated on creating a common language for game design. After comparing the three frameworks, it could be simplified as follows: the Alexander's design language is theory and design method for planning buildings in a strictly defined

paradigm,while software design pattern language is a collection of helpful tools for developers and game design pattern language is a dictionary for comprehension and discussing the formal aspects of gameplay.

The remaining questions are: how can we improve game design patterns in order to make it useful for game designers? Is it valuable and practical method for game design and should be further developed or abandoned? Joris Dormans in his paper *Make Design Patterns Work*[39] confronts both of those problems. He proposes four guidelines that future game design patterns creators should follow: pattern language should have an ideal exemplar of what it means to be a good game, patterns should be insightful and informative rather than just descriptive, patterns should be presented as problem-solution pairs, and relations between patterns should be flexible, as well as plentiful to accommodate different kinds of games[40]. To some extent Dormans' observations and opinions are accurate. He states that the problem-solution structure of a pattern makes the language a prescriptive. Without this distinction of quality in game design we cannot hope for design patterns to be anything more than a collection of definitions. This of course forces the first guideline – the creation of the idea of a model game. Game design is a vast discipline containing both rules for making engaging experience in tile-matching puzzle game like *Bejeweled* (PopCap Games, 2001) and advice for creating believable military simulations like *America's Army* (United States Army, 2002). Therefore, it is very hard and not efficient to create an ideal exemplar for all those games; a better approach would be to build this kind of model for a specific genre or type of game. Game design patterns made for one specific area of design could be much more practical in use, as most video games of the same genre share similar or the same design goals. Thus, rather than creating an enormous library of hundreds patterns for all video games researchers should concentrate on writing them for specific types of games. Furthermore, this method could be adapted by game development studios or publishers specializing in one genre or even one franchise of games, like Ubisoft with their *Assasin's Creed* series (2007-2015) or Inifinity Ward, Treyarch and Sledgehammer Games studios responsible for *Call of Duty* series. For sure similar methods and documents are being produced in those companies, but it does not exclude the possibility for game researchers to support their pursuits with their academic publications. The comparison of pattern languages leads to the conclusion that while there is little sense in writing general language of design patterns, we should strive to make our particular, specific, local languages. As a final comment I propose a paraphrased quotation of Alexander's thought – "In a healthy *game culture* there will be as many patterns languages as there are *games* - even though these languages are shared and similar"[41].

---

39 J. Dormans, *Make Design Patterns Work*, 2013 [on-line:] http://www.jorisdormans.nl/pdf/making-design-patterns-work-dormans.pdf [30.04.2016]

40 Ibidem, p. 4.

41 C. Alexander, S. Ishikawa, M. Silverstein, op. cit., p. xvi.

Bibliography

Alexander C., Ishikawa S., Silverstein M., *A Pattern Language: Towns, Buildings, Construction*, New York 1977.

Alexander C., *The Oregon Experiment,* New York 1975.

Alexander C., *The Timeless Way of Building,* New York 1979.

Beck K., Cunningham W., *Using Pattern Languages for Object-Oriented Programs,* Object-Oriented Programming, Systems, Languages & Applications conference 1987 [on-line:] http://c2.com/doc/oopsla87.html [30.04.2016].

Björk S., Holopainen J., *Patterns in Game Design*, Hingham 2005.

Chen S., Brown D., *The Architecture of Level Design*, 2001 [on-line:] http://www.gamasutra.com/resource_guide/20010716/chen_pfv.htm [30.04.2016].

Church D., *Formal Abstract Design Tools*, 1999 [on-line:] http://www.gamasutra.com/view/feature/131764/formal_abstract_design_tools.php [30.04.2016].

Dormans J., *Engineering Emergence. Applied Theory for Game Design*, 2012 [on-line:] http://www.jorisdormans.nl/pdf/dormans_engineering_emergence.pdf [30.04.2016].

Dormans J., *Make Design Patterns Work*, 2013 [on-line:] http://www.jorisdormans.nl/pdf/making-design-patterns-work-dormans.pdf [30.04.2016].

Dovey K., *The Pattern Language and its Enemies*, "Design Studies" 1990, vol. 11.

Erickson T., *Lingua Francae for Design: Sacred Places and Pattern Languages*, "Proceedings of the ACM Conference on Designing Interactive Systems 2000" New York 2000 [on-line:] http://www.pliant.org/personal/Tom_Erickson/LinguaFranca_DIS2000.html [30.04.2016].

Gamma E., Helm R., Johnson R., Vissides J., *Design patterns: Elements of reusable object-oriented software*, Reading 1995.

Hannemann J., Kiczales G., *Design pattern implementation in Java and AspectJ,* "ACM Sigplan Notices" 2002, vol. 37, no. 11.

Hullett K., Whitehead J., *Design Patterns in FPS Levels* [in:] *Proceedings of Foundations of Digital Games Conference,* Montreal 2010.

Juul J., *A casual revolution: Reinventing video games and their players,* Cambridge 2009.

Koster R., *Grammar of Gameplay. Game Atoms: Can Games Be Diagrammed?*, 2005 [on-line:] http://www.raphkoster.com/gaming/atof/grammarofgameplay.pdf [30.04.2016].

Kreimeier B., *The Case for Game Design Patterns*, 2002 [on-line:] http://www.gamasutra.com/view/feature/132649/the_case_for_game_design_patterns.php?print=1 [30.04.2016].

Lankoski P., *Character-Driven Game Design – A Design Approach and Its Foundations in Character Engagement,* Jyväskylä 2010.

Norvig P., *Design Patterns in Dynamic Programming,* 1996 [on-line:] http://www.norvig.com/design-patterns/design-patterns.pdf [30.04.2016].

Salen K., Zimmerman E., *Rules of play: Game design fundamentals,* Cambridge 2003.

Salingaros N. A., *The Structure of Pattern Languages*, "Architectural Research Quarterly" 2000 , vol. 4, no. 2.

Sullivan L., *The Tall Office Building Artistically Considered,* "Lippincott's Monthly Magazine", March, 1896.

Turner T., *City as landscape: A Post Post-Modern View of Design and Planning*, Abingdon 1996.

Venners B., *Exploring with Wiki. A Converstaion with Ward Cunningham, Part I,* 2003 [on-line:] http://www.artima.com/intv/wiki.html [30.04.2016].

Zagal J. P., Björk S., Lewis C., *Dark patterns in the design of games*, 2013 [on-line:] http://soda.swedish-ict.se/5588/1/DarkPatterns.1.1.6_cameraready.pdf [30.04.2016].

Zagal J. P., Mateas M., Fernandez-Vara C., Hochhalter B., Lichti N., *Towards an Ontological Language for Game Analysis* [in:] *Changing Views: Worlds in Play, Selected Papers of DIGRA 2005*, ed. S. de Castell and J. Jenson, Vancouver 2005.

Abstract

Game design patterns from the university to design studio

In 1977 Austrian architect Christopher Alexander published his best known book *A Pattern Language*. The concept presented nearly 40 years ago inspired computer scientists and programmers alike, leading to the creation of design pattern language for object-oriented programming. Yet, despite the popularity it failed to gain such recognition and acceptance as a practical tool in its own field – architecture. Years later, when the video game industry started to feel the need for a formal language to describe gameplay, an idea to use pattern language appeared. Two game researchers – Staffan Björk and Jussi Holopainen studied and conceived a collection of design patterns. While it inspired game studies circles for some time it did not grab the attention of the wide game industry community.

Following paper examines different concepts of design pattern languages from the original one in architecture, to most popular pattern concept in object-oriented programming and to Björk's and Holopainen's work in the field of video game design. The focal point of analysis and comparison will be the languages foundations and assumptions, single design pattern structure, and level of abstraction. Beside the internal structure of each pattern language, the reception in the professional community will be examined. The central issue addressed in

this paper are the characteristics of specific pattern languages that made them popular and useful or not. After this comparison, both shortcomings of Björk's and Holopainen's concept and possible solutions for similar future theories will be examined.